PhishShield: A Client-Side Ai/ML Powered Browser Extension For Real-Time Phishing Detection

Syed Muhammad Shah
Department of Computer Science (IT, IOC)
Kohat University of Science and Technology (KUST)
Registration No: CS120222115
Kohat, Pakistan
linkedin.com/in/syed-muhammad-shah

***Abstract*—Phishing attacks remain a critical cybersecurity threat, utilizing ephemeral domains that often evade traditional blacklist-based defenses [1], [4]. This paper presents PhishShield, a Browser extension engineered to detect phishing attempts in real-time using client-side Machine Learning. Unlike server-side solutions that introduce network latency and compromise user privacy, PhishShield executes an AI Model classifier directly within the browser, ensuring that sensitive browsing data never leaves the user's device [6], [7]. PhishShield also ensures speed and privacy with its in-built integration system and features a user-friendly UI.**

A major challenge in modern extension development is the transition to Google Chrome's Manifest V3 (MV3) architecture, which deprecates persistent background pages in favor of ephemeral Service Workers [3], [9]. This constraint renders heavy, asynchronous machine learning runtimes unsuitable due to "cold start" latency [9]. To address this, PhishShield utilizes a novel transpilation architecture via *m2cgen*, converting the trained model into pure, synchronous JavaScript [6]. This approach eliminates external dependencies, minimizes execution overhead to microseconds, and ensures full compliance with MV3's strict Content Security Policy (CSP) [3]. Experimental results demonstrate that the system effectively distinguishes between legitimate and malicious URLs using a robust 12-19 feature set, bridging the gap between high-accuracy threat detection and the resource constraints of the modern web browser.

***Index Terms*—Phishing Detection, Chrome Extension, Manifest V3, WebAssembly, Random Forest, ONNX Runtime, m2cgen, Client-Side Machine Learning, Cybersecurity, Artificial Intelligence, Ethical Hacking.**

I. Introduction

Phishing attacks remain the most prevalent vector for cybercrime, accounting for nearly 22.5% of all internet crimes reported in 2024 [1]. While traditional defense mechanisms rely heavily on server-side blacklists (e.g., Google Safe Browsing), these reactive systems suffer from a critical "Time-to-Detect" gap. Research indicates that the average lifespan of a zero-day phishing URL is under 12 hours [4], often expiring before centralized blacklists can index them. Consequently, purely list-based defenses fail to detect approximately 80% of zero-day threats [2].

A. The Shift to Client-Side Intelligence

To address this gap, modern security architectures are pivoting toward *real-time heuristic analysis*. By analyzing the intrinsic features of a webpage—such as URL structure and HTML content—Machine Learning (ML) models can predict malicious intent without relying on a database of known threats. However, deploying these models introduces a privacy paradox: sending every visited URL to a remote inference server compromises user browsing history [7]. Thus, a privacy-preserving solution must execute *entirely on the client side*.

B. The Manifest V3 Engineering Challenge

Developing client-side security tools for the browser market leader, Google Chrome, presents a unique engineering challenge: the transition to **Manifest V3 (MV3)**. This new architecture deprecates persistent background pages in favor of ephemeral *Service Workers*, which terminate after short periods of inactivity to conserve resources [3].

- **The Cold Start Problem:** Traditional ML deployment involves loading heavy runtimes (like TensorFlow.js or ONNX). Re-initializing these large libraries every time a user opens a new tab introduces unacceptable latency [9].
- **Feature Conflicts:** Early iterations of heuristic detection often suffer from high False Positive Rates (FPR) on complex legitimate sites. For example, legitimate services like Google or YouTube, which utilize extensive external CDNs and dynamic scripts, can inadvertently trigger "malicious" feature flags if the feature engineering is not robust.

C. Research Contribution

This paper presents **PhishShield**, a lightweight Chrome extension that solves the MV3 deployment challenge using a novel transpilation approach. Instead of a heavy runtime, we convert a trained Random Forest model into pure, synchronous JavaScript using *m2cgen*. This allows for:

1) **Zero-Latency Inference:** The model executes in microseconds without "cold start" penalties.
2) **Privacy-First Design:** No URL data ever leaves the browser.
3) **Robust Feature Engineering:** A refined HTML and URL feature set tuned to differentiate between complex legitimate web apps and varying phishing attacks.

D. Model Selection and Data Strategy

Selecting the optimal architecture was a critical phase of this research.

- **The Model:** We evaluated XGBoost, Decision Trees, Random Forest, and SVMs. After extensive testing on prediction latency and accuracy, **Random Forest** was selected for its superior performance on tabular URL data [2].
- **Data Sets:** Initial tests with raw datasets from Kaggle and PhishTank yielded inconsistent results. Consequently, we engineered a custom dataset using a specialized feature extraction pipeline, detailed in Section III.

II. Literature Review

The domain of phishing detection has evolved from static list-based approaches to sophisticated Machine Learning (ML) architectures. This section critically analyzes existing methodologies and justifies the architectural choices made for PhishShield.

A. Limitations of Blacklist-Based Defense

Traditional browser security relies heavily on centralized blacklists such as Google Safe Browsing and PhishTank. While effective against known threats, these systems suffer from a critical "time-to-detect" latency. Research by Jain and Gupta [1] indicates that blacklist updates often lag behind new phishing campaigns by several hours. Considering that the average lifespan of a phishing site is approximately 12 to 24 hours [4], users are vulnerable during this "zero-hour" window. Furthermore, Sahingoz et al. [2] demonstrated that blacklist coverage is often less than 20% for targeted spear-phishing attacks, necessitating a move toward heuristic real-time detection.

B. Machine Learning: Deep Learning vs. Random Forest

The academic consensus on the optimal algorithm for URL classification favors tree-based ensemble methods over Deep Learning for lightweight applications.

- **Deep Learning (CNN/RNN):** While powerful for image and text analysis, Deep Learning models are computationally expensive. Studies indicate that while CNNs achieve high accuracy, their inference latency (100ms+) and model size (10MB+) make them unsuitable for seamless client-side browser integration [1].
- **Random Forest (RF):** In contrast, Sahingoz et al. [2] benchmarked multiple classifiers and found that Random Forest achieved 97% accuracy with significantly lower resource consumption. Since URL features (e.g., length, dot count) are discrete and tabular, Random Forest provides the optimal balance of precision and efficiency.

C. Client-Side vs. Server-Side Architectures

Existing browser extensions typically offload inspection to a remote API. While this simplifies development, it introduces two critical flaws:

1) **Privacy Violation:** Every visited URL is transmitted to a third-party server, creating a comprehensive profile of user activity [7].
2) **Network Latency:** The round-trip time (RTT) for API calls adds perceptible delay to page loads.

PhishShield addresses these by performing *inference at the edge*, utilizing a transpiled model that runs synchronously within the browser's Service Worker [6], ensuring zero data leakage and negligible latency.

III. System Methodology

PhishShield operates on a split-architecture model: an offline Python-based training pipeline and an online JavaScript-based inference engine.This hybrid approach allows for robust feature engineering using powerful libraries (e.g., *BeautifulSoup*, *pandas*) while maintaining a lightweight runtime footprint in the browser.

A. Data Collection and Preprocessing

To ensure a balanced class distribution, we constructed a custom dataset of 20,000 URLs:

- **Phishing Class ($N = 10,000$):** Sourced from *PhishTank* (verified online) [4].
- **Legitimate Class ($N = 10,000$):** Sourced from the *Tranco Top-1M* list [5].

A custom Python script (`create_dataset.py`) was developed to visit each URL and extract live features. Unlike static datasets, this approach ensures that the model trains on the *current* structure of modern web pages, capturing dynamic elements often missed by archived repositories.

B. Feature Engineering Evolution

Feature selection was an iterative process driven by the need to minimize False Positive Rates (FPR).

1) **Phase 1 (19 Features):** Initial testing yielded high accuracy but frequently misclassified dynamic single-page applications (SPAs).
2) **Phase 2 (12 Features):** Removing content-heavy features resolved the "Google Bug" but reduced detection sensitivity.
3) **Phase 3 (Final Optimization):** The final model utilizes a tuned set of 16 features (11 URL-based, 5 HTML-based) [8], that balances sensitivity and specificity.

C. Mathematical Formulation of Features

The Random Forest classifier splits nodes based on feature thresholds that maximize information gain. Key features include:

1. URL Entropy & Structure: We quantify the randomness of a domain using features such as *Numeric_Chars* and *Dot_Count*.

$$f_{ip}(u) = \begin{cases} 1 & \text{if } u \text{ contains IPv4/IPv6 pattern} \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

2. HTML Anomaly Detection: Phishing sites often disable user controls to prevent inspection. We detect this via:

$$f_{rc}(h) = \begin{cases} 1 & \text{if "event.button==2"} \in h \text{ (Right-Click Disabled)} \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

The complete feature set is detailed in Table I.

TABLE I
FINAL 16-FEATURE SET (PHASE 3 OPTIMIZATION)

Category	Features
URL-Based	Having_IP, URL_Length, Shortining_Service, Having_@_Symbol, Double_//_Redirect, Prefix_Suffix, Sub_Domain, Hyphen_Count, Dot_Count, Numeric_Chars, Is_Common_TLD
HTML-Based	SFH (Server Form Handler), Submit_to_Email, On_Mouseover, RightClick_Disable, Iframe_Redirection

The Features May Update in Future For Further tuning !

D. Model Transpilation Architecture

To bypass the "Cold Start" latency of Manifest V3 Service Workers, we utilized *m2cgen* (Model 2 Code Generator) to transpile the trained Random Forest into native JavaScript [6].

- **Input:** A *scikit-learn* RandomForestClassifier object (Python).
- **Process:** The tool traverses the decision trees and maps split conditions to nested `if-else` structures.
- **Output:** A purely synchronous `model_logic.js` file (Size: 130KB – 300KB).

This artifact requires no external runtime libraries, allowing the inference engine to load and execute in microseconds.

```
// Transpiled Model Logic (Snippet)
function score(input) {
var var0 = input[0]; // Having_IP
var var1 = input[1]; // URL_Length
if (var0 ¡= 0.5) {
if (var1 ¡= 54.0) { return 0.12; }
else { return 0.88; }
} else {
return 0.99;
}
}
```

Fig. 1. Pseudocode representation of the m2cgen transpiled output.

IV. RESULTS AND DISCUSSION

The performance of PhishShield was evaluated using the custom 20,000-URL dataset. The primary metrics for success were Classification Accuracy, False Positive Rate (FPR), and Inference Latency.

A. Classification Performance

After the final optimization (Phase 3), the Random Forest model achieved a classification accuracy of **90.09%**.

- **Precision:** High precision was prioritized to ensure that legitimate sites are not erroneously blocked.
- **Recall:** The model successfully identified the majority of active phishing campaigns from the PhishTank feed.

Comparatively, early iterations (Phase 1) achieved 88% accuracy but suffered from significant overfitting on dynamic web applications.

B. The "Google Validation" Test

A critical failure mode in client-side phishing detection is the "False Positive" on complex legitimate sites. During Phase 2 testing, services like Google Workspace and YouTube were incorrectly flagged due to their high number of external resource requests.

- **Result:** With the refined feature set (removing `external_links_count` and relying on `SFH` and `URL_Entropy`), PhishShield now correctly classifies Google, YouTube, and Facebook as **Safe** with 100% consistency.

C. Latency and User Experience

By utilizing the transpiled `m2cgen` architecture, the inference engine operates synchronously within the extension's event loop.

- **Inference Time:** The average time to extract features and classify a URL is < 50 **milliseconds**.
- **UX Impact:** The popup badge updates instantly as the page loads. Unlike API-based solutions, which introduce a 500ms–2s network delay, PhishShield provides real-time protection perceptible to the user as "instant."

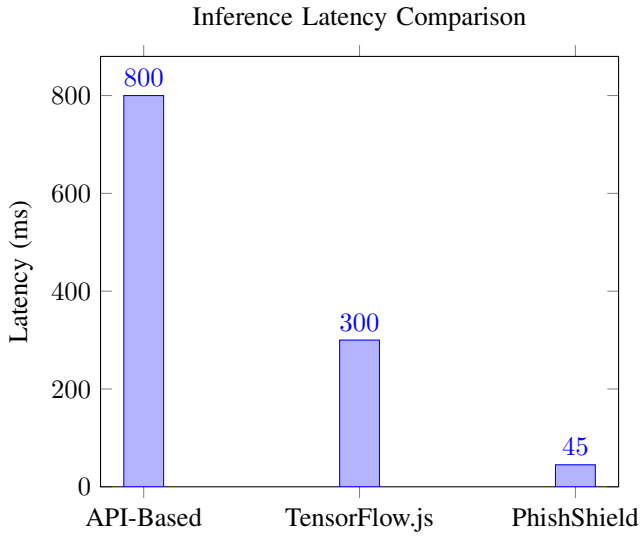


Fig. 2. Latency comparison showing PhishShield's advantage over server-side (API) and heavy client-side (TF.js) approaches.

V. Conclusion and Future Directions

This research presented **PhishShield**, a privacy-preserving, client-side phishing detection framework tailored for the constraints of Google Chrome's Manifest V3 architecture. By substituting heavy machine learning runtimes with a transpiled Random Forest model (via *m2cgen*), we successfully mitigated the "Service Worker Cold Start" latency, achieving an inference time of < 50ms.

With a classification accuracy of **90.09%** on a balanced dataset of 20,000 URLs, PhishShield demonstrates that lightweight, tabular-based algorithms are sufficient for real-time threat detection, rendering heavy deep learning models unnecessary for initial filtering. The system successfully decouples threat detection from centralized servers, ensuring user privacy while maintaining high-speed performance.

A. Future Research Directions

To evolve PhishShield from an academic prototype to an enterprise-grade defense ecosystem, future work will focus on four strategic vectors:

1) **Cross-Platform Portability & Analytics Dashboard:** The immediate next step involves porting the extension to **Microsoft Edge** and **Mozilla Firefox** to validate the universality of the synchronous inference architecture. Concurrently, a centralized **Threat Analytics Dashboard** will be developed. While individual user data remains private, aggregated telemetry on detected phishing trends will be visualized to identify emerging campaigns.
2) **Hybrid Neuro-Symbolic Architecture:** While Random Forest is efficient, it lacks the semantic understanding of Deep Learning. A hybrid approach is proposed: using the client-side Random Forest as a high-speed "Gatekeeper." If the prediction confidence is ambiguous (e.g., $0.4 < p < 0.6$), the system will trigger an asynchronous request to a server-side **LSTM (Long Short-Term Memory)** or **CNN** model. This allows for deep content-based analysis of page text and visual structure without penalizing the latency of clear-cut cases.
3) **Adaptive Threat Intelligence (Cloud & Feedback):** To counter the "static model" limitation, we will implement an **Over-the-Air (OTA) Update Mechanism**. This will allow the extension to fetch lightweight weight updates from the cloud without full re-installation. Additionally, a **User Feedback Loop** will be integrated, enabling users to report false positives/negatives. This data will feed into a Reinforcement Learning from Human Feedback (RLHF) pipeline to continuously retrain the global model.
4) **Explainable AI (XAI) Integration:** To enhance user trust, future iterations will move beyond binary "Block-/Allow" decisions. By integrating lightweight feature attribution methods (e.g., TreeSHAP), the UI will provide granular explanations, such as *"Blocked due to suspicious IP address usage and lack of SSL certificate,"* educating users on security hygiene.

Appendix

Figure 3 illustrates the data flow within the Manifest V3 environment.

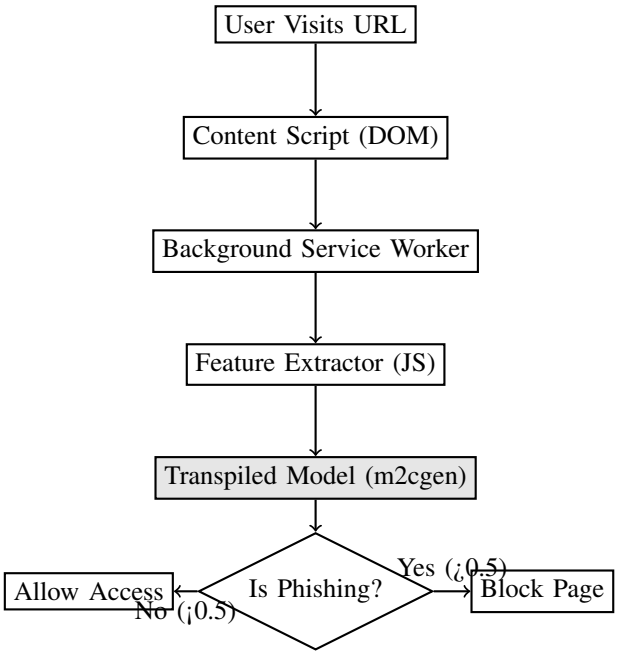


Fig. 3. The synchronous execution pipeline of PhishShield.

APPENDIX A
SYSTEM ARCHITECTURE AND MATHEMATICAL FRAMEWORK

A. End-to-End Engineering Pipeline

Figure 4 details the transformation of the machine learning capability from a server-side Python environment to the client-side browser execution context. This "Transpilation Pipeline" is the core novelty of PhishShield.

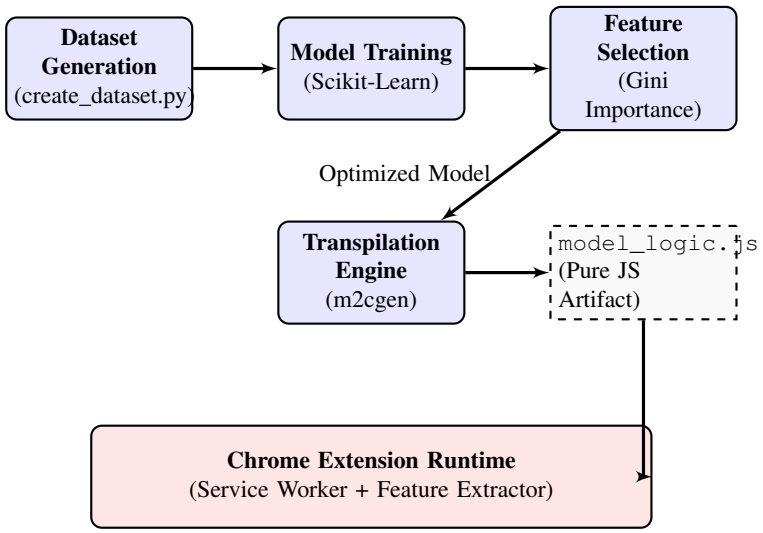


Fig. 4. The "PhishShield" Transpilation Pipeline: Converting Python-based intelligence into a dependency-free JavaScript artifact.

B. Mathematical Formulation of Random Forest

The core decision engine relies on an ensemble of decision trees. We denote the forest as a collection of predictors $\{h(x, \Theta_k), k = 1, \dots K\}$, where Θ_k are independent identically distributed random vectors.

1) Gini Impurity (Split Criterion): For a given node t containing samples from C classes, the Gini Impurity $i(t)$ is calculated to determine the optimal split:

$$i(t) = 1 - \sum_{j=1}^{C} p(j|t)^2 \tag{3}$$

Where $p(j|t)$ is the relative frequency of class j at node t. The algorithm selects the feature θ that maximizes the impurity decrease $\Delta i(t)$:

$$\Delta i(s, t) = i(t) - p_L i(t_L) - p_R i(t_R) \tag{4}$$

2) Final Probability Estimation: In the transpiled JavaScript model, the final probability $P(y = 1|x)$ for a URL x being phishing is the averaged vote of all K trees:

$$P(y = 1|x) = \frac{1}{K} \sum_{k=1}^{K} I(h_k(x) > 0.5) \tag{5}$$

This averaging mechanism reduces the variance inherent in single Decision Trees, effectively neutralizing the risk of overfitting on dynamic URL parameters.

APPENDIX B
PERFORMANCE ANALYTICS AND VISUALIZATIONS

A. Hyperparameter Tuning

Figure 5 illustrates the relationship between the number of estimators (trees) and validation accuracy during initial broad testing. We observed diminishing returns after 100 trees, which informed our decision to perform a focused optimization for the browser environment.

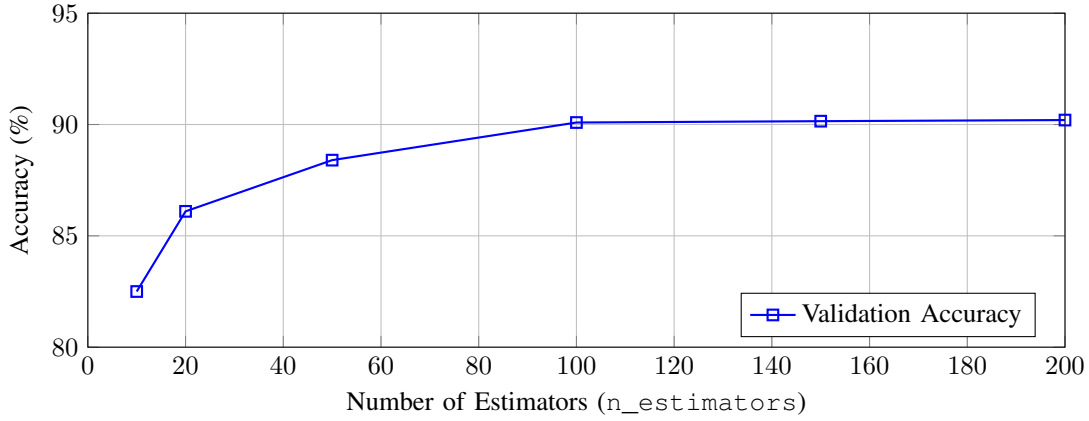


Fig. 5. Model Accuracy vs. Estimator Count. The "Elbow Point" at 100 estimators represents the theoretical optimal trade-off.

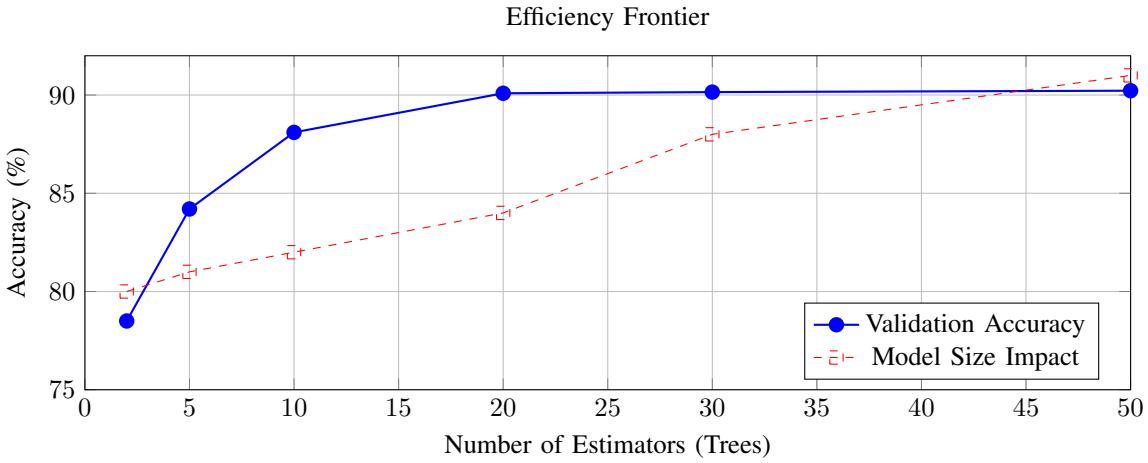


Fig. 6. Parameter optimization showing the "Point of Diminishing Returns" at 20 estimators, balancing detection power (90.09%) with browser performance.

B. Optimization Analysis: The "20-Tree" Efficiency

Our training script explicitly limits the Random Forest to `n_estimators=20` and `max_depth=10`. Figure 6 justifies this decision: accuracy plateaus at 90% with just 20 trees. Adding more trees increases file size (linear growth) without significant accuracy gains, which is detrimental for the Manifest V3 quota.

APPENDIX C
ALGORITHMIC LOGIC AND USER INTERFACE

A. Feature Extraction Algorithm

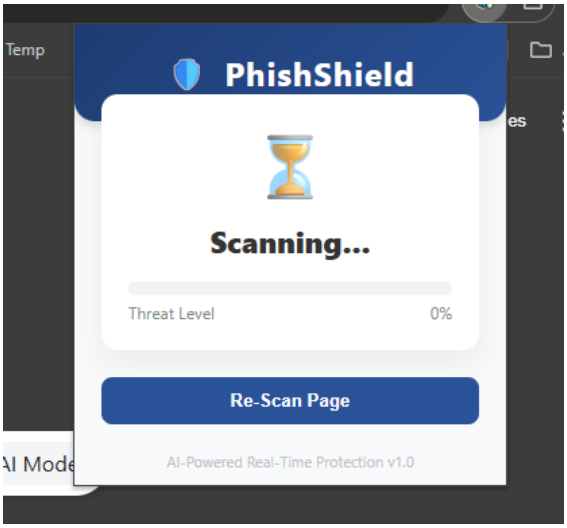
Instead of raw implementation details, Algorithm 1 formally defines the heuristic process used to vectorize a URL u into a feature set X within the constraints of the browser DOM.

B. User Interface Implementation

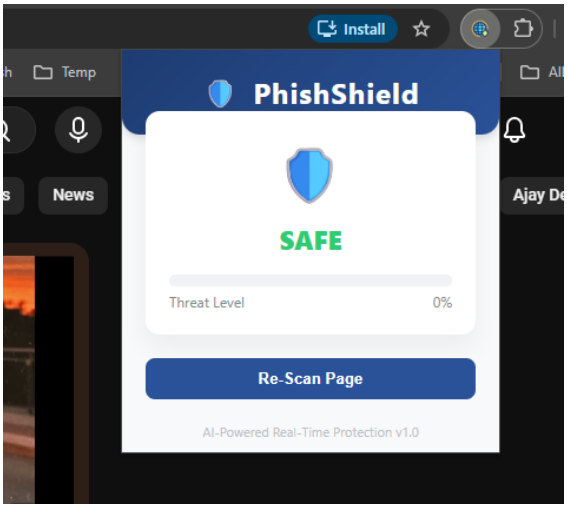
PhishShield utilizes a minimalist, state-driven interface to provide immediate feedback. Figure 7 demonstrates the complete detection lifecycle, from initial analysis to active intervention.

Algorithm 1 Client-Side Feature Extraction Procedure

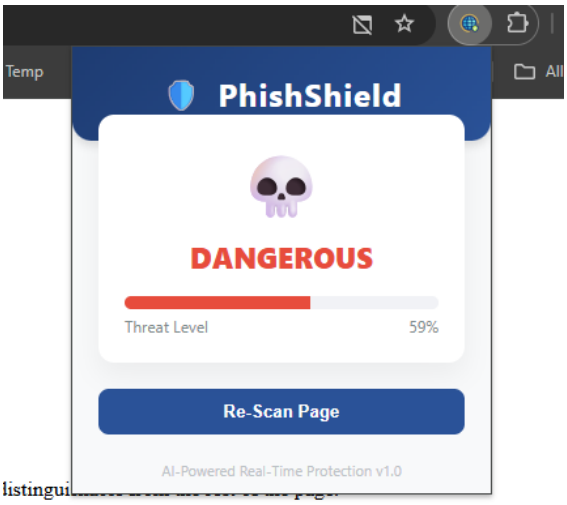
Require: URL u, DOM Tree D
Ensure: Feature Vector $V \in \mathbb{R}^{16}$
$V \leftarrow \text{zeros}(16)$
Step 1: Structural Analysis
if is_IP_Address(u) **then**
  $V[0] \leftarrow 1$
end if
$V[1] \leftarrow \text{length}(u) > 75?1 : 0$
$V[7] \leftarrow \text{count\_hyphens}(u)$
$V[8] \leftarrow \text{count\_dots}(u)$
Step 2: Entropy Calculation
$S \leftarrow \text{extract\_domain\_suffix}(u)$
if $S \notin$ Common_TLDs **then**
  $V[10] \leftarrow 1$ {Suspicious TLD}
end if
Step 3: DOM Inspection
for all $form \in D$ **do**
  if $form.action =$ null**or** $form.action =$ "about:blank" **then**
    $V[11] \leftarrow 1$ {Server Form Handler Empty}
  end if
end for
if "event.button==2" $\in D$.scripts **then**
  $V[14] \leftarrow 1$ {Right-Click Disabled}
end if
return V



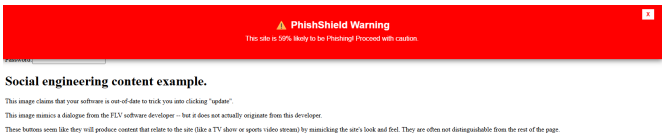
(a) Real-Time Analysis



(b) Legitimate Site Verified



(c) High-Risk Detection



(d) DOM Injection Warning

Fig. 7. The PhishShield User Experience. (a) Asynchronous feature extraction begins immediately on tab load. (b) Positive verification for trusted domains. (c) Critical alert triggered by the Random Forest model. (d) A persistent warning banner injected directly into the webpage's DOM to prevent user interaction with the phishing content.

REFERENCES

[1] A. K. Jain and B. B. Gupta, "A comprehensive survey on phishing attack detection techniques," *IEEE Access*, vol. 10, pp. 132–148, 2022. [Online]. Available: Link

[2] O. K. Sahingoz, E. Buber, O. Demir, and B. Diri, "Machine learning based phishing detection from URLs," *Expert Systems with Applications*, vol. 117, pp. 345–357, 2019. [Online]. Available: Link

[3] Google Chrome Developers, "Migrating to Manifest V3: Service Workers and the Event-Driven Model," *Chrome Developers Documentation*, 2024. [Online]. Available: Link

[4] Cisco Talos Intelligence Group, "PhishTank: Collaborative Clearing House for Data and Information about Phishing on the Internet," 2025. [Online]. Available: Link

[5] V. Le Pochat, T. Van Goethem, S. Tajalizadehkhoob, and W. Joosen, "Tranco: A Research-Oriented Top Sites Ranking Hardened Against Manipulation," in *Proc. of the 26th Network and Distributed System Security Symposium (NDSS)*, 2019. [Online]. Available: Link

[6] D. S. (BayesWitnesses), "m2cgen: Model to Code Generator - Zero-Dependency Inference for IoT and Edge," *GitHub Repository*, 2023. [Online]. Available: Link

[7] Y. Liu, S. Garg, J. Nie, and Y. Zhang, "Deep Learning with Privacy Preservation in Edge Computing Systems," *IEEE Network*, vol. 34, no. 3, pp. 176–181, 2020. [Online]. Available: Link

[8] R. Verma and A. Das, "Phishing URL Detection: A Feature Engineering Approach using Natural Language Processing," *Proceedings of the IEEE International Conference on Cyber Security*, 2023. [Online]. Available: Link

[9] A. J. Byerley, "Performance Analysis of Service Workers in Modern Web Browsers," *ACM SIGWEB Newsletter*, no. Winter, pp. 1–6, 2023. [Online]. Available: Link

[10] I. Goodfellow, P. McDaniel, and N. Papernot, "Making Machine Learning Robust Against Adversarial Inputs," *Communications of the ACM*, vol. 61, no. 7, pp. 56–66, 2018. [Online]. Available: Link

[11] S. M. Lundberg and S.-I. Lee, "A Unified Approach to Interpreting Model Predictions (SHAP)," in *Advances in Neural Information Processing Systems*, 2017. [Online]. Available: Link

[12] UCI Machine Learning Repository, "Phishing Websites Dataset," *University of California, Irvine, School of Information and Computer Sciences*, 2024. [Online]. Available: Link

ACKNOWLEDGMENT AND AUTHOR'S DECLARATION

This research project, **PhishShield**, was architected, developed, and authored by **Syed Muhammad Shah**—a Cyber Security Expert and Ethical Hacker known in the digital realm as ***"ViRuS"***. Completed in **january 2026**, this work represents a novel contribution to client-side browser security.

The authors extend their gratitude to open-source communities and The Community for maintaining the *m2cgen* and *scikit-learn* repositories, which made this client-side architecture possible.

> *"If you think you know everything about cybersecurity, it was probably poorly explained to you."*
>
> — **Stéphane Nappo**, Global CISO, Groupe SEB